Simulations with VRML and Dynast

Antonín Slavík

Simulations with VRML and Dynast

by Antonín Slavík

Table of Contents

1. Demodyn applet - User's Guide 1
1.1. Running the applet 1
1.2. Applet user interface 2
1.3. Running demodyn from local disk 4
2. VRML Editor - User's Guide 5
2.1. Running the editor 5
2.2. Elementary shapes 6
2.3. Setting shape appearance 7
2.4. Transformation nodes 9
2.5. Putting VRML and Dynast together 11
2.6. Using clipboard and links 13
2.7. Viewpoints 15
2.8. Prototypes 16
2.9. Importing VRML nodes 21
2.10. Using offline version 21

List of Figures

1-1. HTML page with applet and VRML browser 1
1-2. Demodyn user interface 2
1-3. Precalculate simulation dialog 2
1-4. Setting parameter values 3
2-1. Editor’s main panel 5
2-2. Dialog Save / Load 6
2-3. Entering node name 6
2-4. Shape settings dialog (1) 7
2-5. Shape settings dialog (2) 7
2-6. Emissive, diffuse and specular colour 8
2-7. Sphere with a texture 9
2-8. Transform settings dialog 9
2-9. Door model 11
2-10. Dynast variables dialog 12
2-11. Viewpoint settings dialog 15
2-12. Moon and Earth simulation 15
2-13. Selecting new prototype 17
2-14. Car prototype settings 17
2-15. Scene with road and two cars 18
2-16. Label prototype settings 19
2-17. Initial conditions in demodyn applet 20

Chapter 1. Demodyn applet - User's Guide

The *demodyn* applet is responsible of viewing simulations created using Dynast and VRML. The applet communicates (using a TCP/IP protocol) with the Dynast server, which computes the simulation results. The applet sends these results to the VRML browser, which writes them into VRML nodes' parameters.

1.1. Running the applet

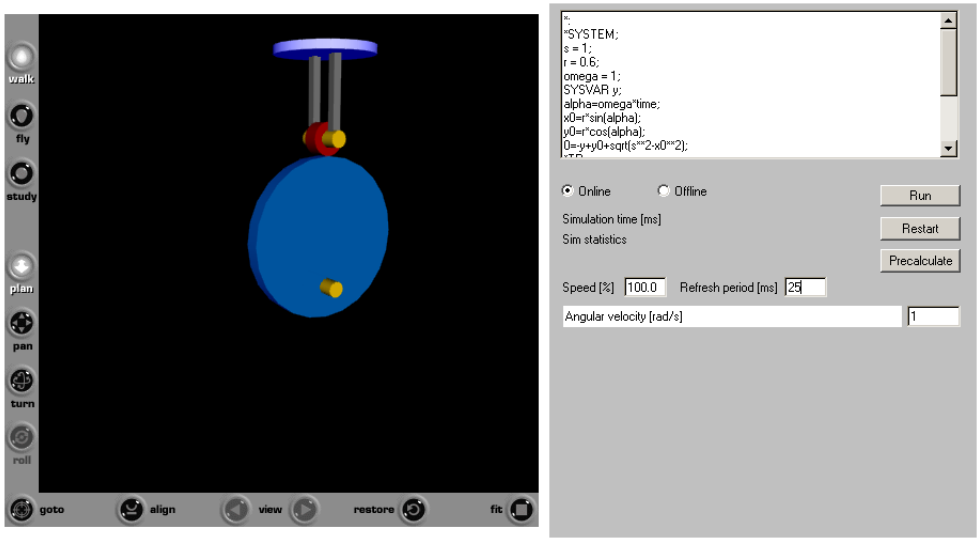
Here are the requirements for viewing simulations using demodyn:

- web browser with Java support, recommended MS Internet Explorer 6.0 or newer
- VRML plugin, recommended Cortona VRML Client 4.1 or newer, can be downloaded at Parallel Graphics (http://www.parallelgraphics.com)
- Internet connection
- fast computer (frequency at least 500 MHz, recommended 1 GHz and a 3D-accelerated graphics card)

Under normal circumstances a Java applet can communicate only with its home server. Because demodyn needs to cooperate with the Dynast server, it has to be placed (and the simulations, too) at the same server, i.e. `virtual.cvut.cz`. In the following text we suppose you are using simulations from this server. An alternative way (viewing simulations from local disk) is discussed in section Running demodyn from local disk.

The simulations are available as HTML pages with embedded demodyn applet, which is run automatically after opening the page with a web browser.

Figure 1-1. HTML page with applet and VRML browser



There is a VRML browser window showing the 3D model in the left part of the screen. We suppose the reader is already familiar with the browser's interface (help is available in the context menu). You can rotate or zoom the view at any time during the simulation.

Note: If you don't see the VRML browser window it probably means that the browser was not correctly installed and you should reinstall it.

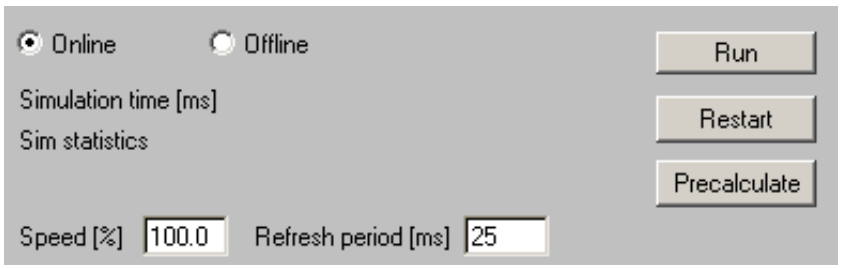
The right part contains the applet's user interface, which is discussed in the following section.

Note: In case of problems make sure that you enabled Java applets in your browser. If you installed Java Runtime Environment from Sun, switch it off and use Microsoft Virtual Machine. The problem can be often identified by opening the Java console.

1.2. Applet user interface

We start with the decription of the components in the middle part of the panel. There are the Run, Restart and Precalculate buttons, the Online/Offline switch and Speed and Refresh period textfields:

Figure 1-2. Demodyn user interface



The Run button starts the simulation (after establishing the connection to Dynast server). The button's label changes to Pause - clicking it pauses the simulation, clicking it once more continues the simulation. The Restart button start simulation from the beginning.

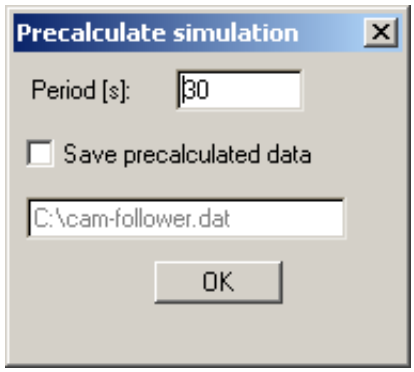
Note: If the simulation doesn't start after pressing Run it is probably due to a network error when attempting to connect to Dynast server. Check out the Java console for more details.

The playing speed can be set using the Speed textfield. Real-time playing corresponds to 100% and e.g. 50% slows down to half speed, 300% means three times faster.

The Refresh period value is the animation step length in miliseconds (i.e. the time between displaying two successive frames). The default value 25 ms corresponds to 40 frames per second. To achieve such a smooth simulation you need a fast Internet connection to the Dynast server. The Sim statistics area displays useful information which help you set the Refresh period correctly: The delay value is the time between displaying the last two frames and in an ideal case it would be equal to the Refresh period (it is higher in case of slow connection). The avg, min and max labels display the average, minimum and maximum of all delay values since the simulation beginning, respectively.

The simulation becomes jerky in case of slow connection. It is then reasonable to use the so-called offline mode: We first request the data from Dynast server and store the results in computer's memory. These results are then used in the second stage, i.e. when viewing the simulation. Click the Precalculate button and enter the number of seconds to precalculate:

Figure 1-3. Precalculate simulation dialog



Leave the Save precalculated data option turned off - it will be explained in the last section. The computation starts after clicking OK.

1. The computation length depends on the *Refresh period*, too (use larger values for slowly varying simulations). Too long computation may be aborted by reloading the page in browser (refresh button).
2. The *Speed* value also influences the computation. Assume you enter *Speed*=200%, *Refresh period*=25 ms and compute the first 30 seconds of the simulation. The result is a 30 second long simulation with 25 ms animation step (i.e. 1 minute of the original simulation with 50 ms step). However the *Speed* value is also used when playing the precomputed simulation: If you play it at 100% (this is the value set after precalculating), you get a 30 second simulation. If you set e.g. *Speed*=200%, you get a 15 second simulation, which runs four times faster than the original simulation (the speed factors multiply together).

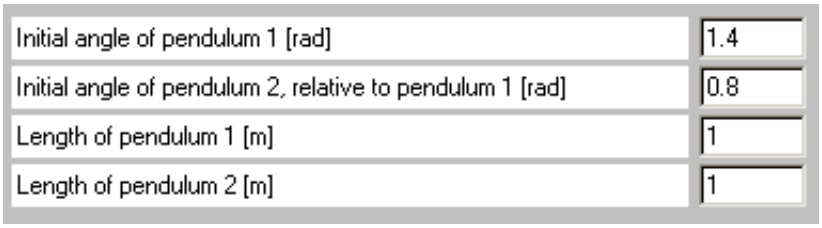
The applet switches to offline mode after the computation finished (it selects Offline instead of Online). The precalculated simulation can be played in the usual way, i.e. using the buttons Run/Pause and Restart. The simulation ceases running after the period you entered in the Precalculate simulation dialog.

We didn't yet mention the script area in the upper part of the applet panel. The script is sent to the Dynast server before the simulation begins (after pressing Run for the first time and each time you press Restart). It is possible to modify various constants, initial conditions etc. in the Dynast script. The changes become apparent after restarting the simulation.

> **Note:** In case of syntax error in the Dynast script the simulation won't start and the applet stops responding for a few seconds. After a while it recovers from the error and we can fix the mistake (the error description is written to the Java console).

Some simulations offer the possibility to set parameters in a more comfortable way. The lower part of the panel then lists the parameters' description and their current values:

Figure 1-4. Setting parameter values



Each time you modify a parameter's value confirm it with Enter. The new values either influence the VRML scene or appear in the Dynast script.

> **Note:** It makes no sense to modify the initial values (or even the Dynast script) in offline mode since the simulation is already stored in the computer's memory.

1.3. Running demodyn from local disk

We already mentioned the security limitation which allows Java applet to communicate only with its home server. It is possible to overcome this limitation in case of MS Internet Explorer browser: Go to the Tools menu, select Internet options and Security. Switch to Local intranet and click Custom level. Find the Java permission item, select Custom and click Java custom settings. Turn on all permissions for unsigned content.

Now you may use the demodyn applet from your local disk (it is now able to establish a connection with the remote Dynast server). This means that you don't have to open a simulation from Internet each time if you have a local copy of it. However, you still need Internet connection for running the simulation.

Another advantage is that you can refer to your local files (such as images or prototypes) in the VRML files.

The applet is now able to write files to disk, too. In this way it is possible to save data from precalculated simulations. Select the Save precalculated data option in Precalculate simulation dialog and enter name of the file name to store the data in. Now how do we use the data from such files?

A single simulation consists of four files which have a common base name and differ only in extension - e.g. `sim.htm`, `sim.wrl`, `sim.prb` and `sim.cfg`. If we now save the data to file `sim.dat` in the same directory and open `sim.htm`, the applet will automatically read the data from sim.dat and enables us to play the simulation in offline mode (it switches to offline mode after reading the data file). In this way users can view simulations without having an Internet connection.

The demodyn applet can be configured in yet another way. Have a look at the `demodyn.cfg` file:

```
dynast_server_address=virtual.cvut.cz;
dynast_server_port=3000;
```

In case you have a local Dynast server installed (see http://virtual.cvut.cz for information about obtaining Dynast server), you may replace the address and port in the configuration file with your own data. This forces demodyn to communicate with the new server instead of `virtual.cvut.cz` and you don't need Internet connection any more.

Chapter 2. VRML Editor - User's Guide

The VRML editor has been developed to provide a user-friendly interface for creating simulations in VRML and Dynast. It produces simulations which can be later viewed using the demodyn applet. You can also use it as a standalone VRML editor.

2.1. Running the editor

Here are the requirements for running the VRML editor:

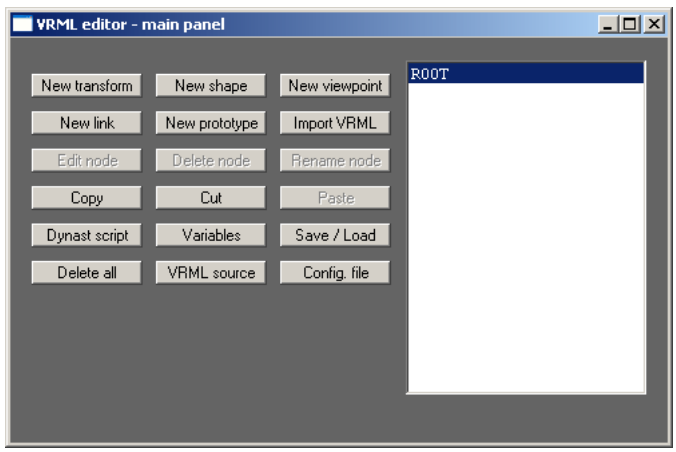
- web browser with Java support, recommended MS Interner Exploret 6.0 or newer
- VRML plugin, recommended Cortona VRML Client 4.1 or newer, can be downloaded at Parallel Graphics (http://www.parallelgraphics.com). We suppose the reader uses this plugin and is already familiar with its interface (help is available in the context menu).
- Internet connection (see the next paragraph)
- fast computer (frequency at least 500 MHz, recommended 1 GHz and a 3D-accelerated graphics card)

Under normal circumstances a Java applet cannot write files to disk. The online version of editor solves the problem in the following way: The simulations are stored using a so-called Dyncad server at `virtual.cvut.cz`. The editor is also situated at `virtual.cvut.cz`, namely at http://as.matfyz.cz/editor/editor.htm or http://virtual.cvut.cz/fm-files/slavik/editor/editor.htm.

In the following text we suppose the reader uses the online version. The offline version can be downloaded from the same adresses in `editor_offline.zip`. This offline version is discussed in the section at the end of the guide.

After opening `editor.htm` in your web browser you should see a VRML browser window and the editor's main panel:

Figure 2-1. Editor's main panel

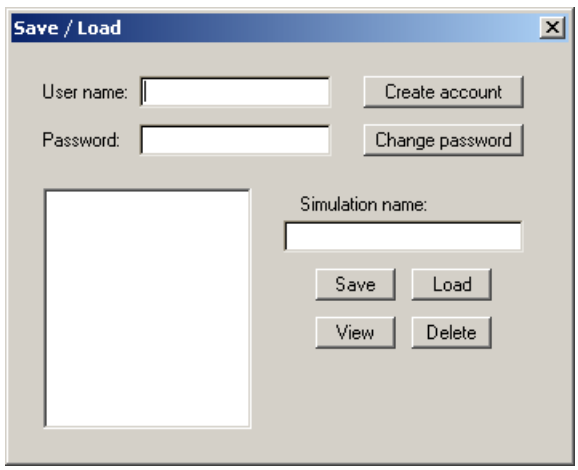


Note: In case of problems make sure that you enabled Java applets in your browser. If you installed Java Runtime Environment from Sun, switch it off and use Microsoft Virtual Machine. The problem can be often identified by opening the Java console.

The main panel contains a group of buttons and a so-called *nodes list* (it contains only ROOT node at the moment). All VRML objects (such as cubes, cylinders or light sources) are called nodes. We explore various node types in the following sections.

The rest of this section explains how to create a user account at the Dyncad server - you will need it for saving and executing simulations. You may skip this step and return here just before saving the first simulation. Click the Save / Load button in the main panel. The following dialog appears:

Figure 2-2. Dialog Save / Load



You have to choose a user name and a password and type them in the corresponding textfields. You will need them each time you save or load a simulation. Click Create account and enter your email address. After establishing the connection with Dyncad server you will be informed whether it created the account successfully. In case an account with the given name already exists you have to choose another one.

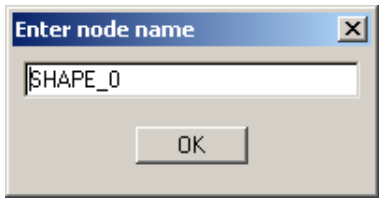
Each time you enter a valid user name and password and confirm it with Enter the editor lists all simulations available on that account (now it is empty). You load a previously saved simulation by selecting it in the list and clicking Load. You can also view the simulation in demodyn applet by clicking the View button.

It is possible to create a so-called public account which can be read by all users without knowing the password (and it can be written to only with the password). Public account names start with the `examples` prefix. There is an account called `examples_vrml` which contains all simulations from this tutorial and a few others. Just type `examples_vrml` in the User name textfield and press Enter. You can view or load all simulations as described in the previous paragraph.

2.2. Elementary shapes

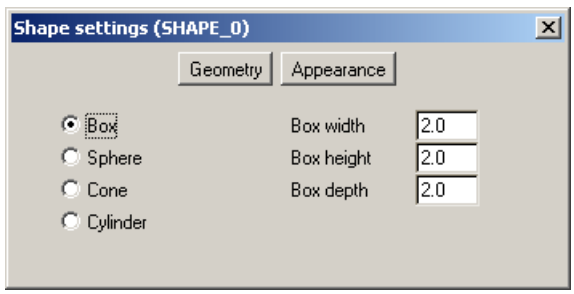
The elementary shapes at our disposal comprise boxes, spheres, cylinders and cones. We select the ROOT node in the nodes list and click the New shape button in the main panel. Each node must be given a name:

Figure 2-3. Entering node name



We use either the default name or enter a new one (such as `WHEEL`, `TANK` etc.). The name must start with a letter and cannot contain spaces. Remember that each node must have a unique name. In case you enter an already existing or invalid name the node won't be created. After clicking `OK` the new node appears in the nodes list and the Shape settings dialog opens:

Figure 2-4. Shape settings dialog (1)



We select shape type in the left part of the dialog. The shape appears immediately in the VRML browser window and we can examine it from different views. Shape dimensions are entered in the right part and are always in metres. New values must be confirmed with Enter.

Almost all labels are self-explanatory, we just mention that in case of cylinder and cone we can disable the shape base or surface by clicking the corresponding checkbox.

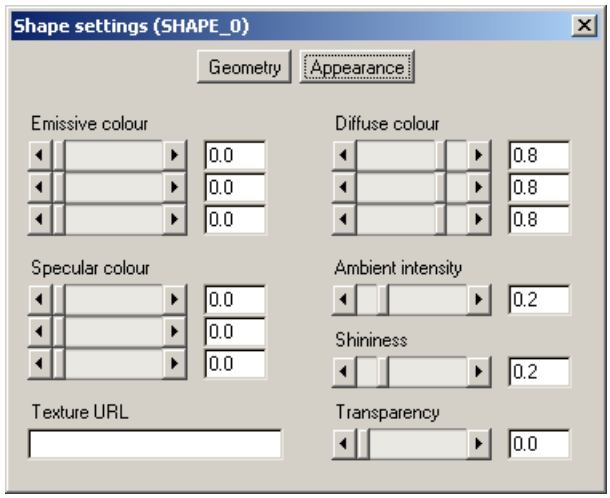
We close the Shape settings dialog in the usual way by clicking the button in the upper-right corner (it is however possible to leave it open and continue editing other nodes). The dialog can be opened later by selecting the node name in the nodes list (the corresponding shape flashes) a pressing the Edit node button. The same action can be performed by double-clicking the node name.

An existing node can be deleted by selecting it in the nodes list and pressing the Delete node button in the main panel (the `ROOT` node cannot be deleted). We can start with an empty scene by using the Delete all button (or by reloading the page in web browser).

2.3. Setting shape appearance

The shapes we created were all grey. We have to use colors to create more impressive simulations. Click the Appearance button in the Shape settings dialog (you can later return to setting shape dimensions by clicking Geometry):

Figure 2-5. Shape settings dialog (2)



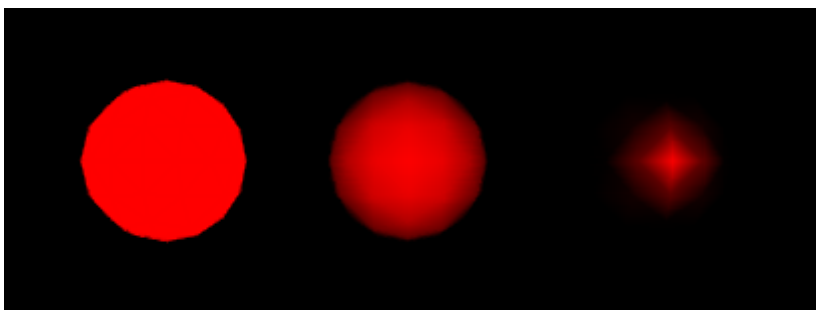
Most computer applications describe colours using the RGB model, i.e. by specifying the amount of red, green and blue colour which should be mixed. In our case we use numbers from [0, 1] interval for the individual R, G, B components.

A VRML object's appearance is described by three parameters (i.e. nine numbers):

- emissive color - it is nonzero in case of objects emitting light. The object colour is constant at all surface points.
- diffuse color - it is nonzero in case of opaque objects. The front side (the one we are looking at) is the brightest, the light intensity decreases as the angle between surface normal and the direction of our view increases.
- specular color - it is nonzero in case of glossy objects. The shape faces perpendicular to the direction of our view reflect the light from our flash-light, but the reflection coefficient decreases rapidly with increasing angle.

Let's test the influence of individual parameters to a shape's appearance: Create a sphere and set all three colours to zero. Then have a look at what happens if you increase the red component of emissive color, diffuse color or specular color.

Figure 2-6. Emissive, diffuse and specular colour



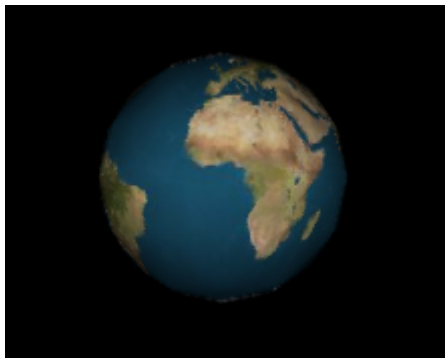
The three types of colour might be freely combined. The right appearance can be most easily found by trial. In case you need a more detailed description of the three parameters consult the The Virtual Reality Modeling Language (http://www.web3d.org/technicalinfo/specifications/eai/index.html) specification. It also explains that the VRML scene nodes can be influenced by other light sources (not only by our flash-light). These sources are not directly available in our editor, but you can insert them using the import procedure (see Importing VRML nodes). The ambient intensity

parameter is also relevant only when using the additional sources. The shininess parameter might be useful in connection with specular colours - it determines the rate of decreasing the light intensity with increasing angle.

We often make use of the transparency parameter. The default shapes are opaque; by increasing the value of this parameter they become more and more transparent.

The most realistic appearance can be achieved with the help of textures - bitmap images (in JPEG or PNG format) which cover the object's surface:

Figure 2-7. Sphere with a texture



The path to the texture file is entered into the Texture URL textfield and is confirmed with Enter. You may use both local files (such as `C:\img\earth.jpg`) and files accessible from Internet (e.g. `http://as.matfyz.cz/examples/earth.jpg`).

> **Note:** In case you work with the online version of editor use only textures accessible from Internet - the simulations are stored at `virtual.cvut.cz` and the textures must be visible from there.

> **Tip:** You can freely combine colours with textures. In this way it is possible to brighten a texture or shift it to a specific hue.

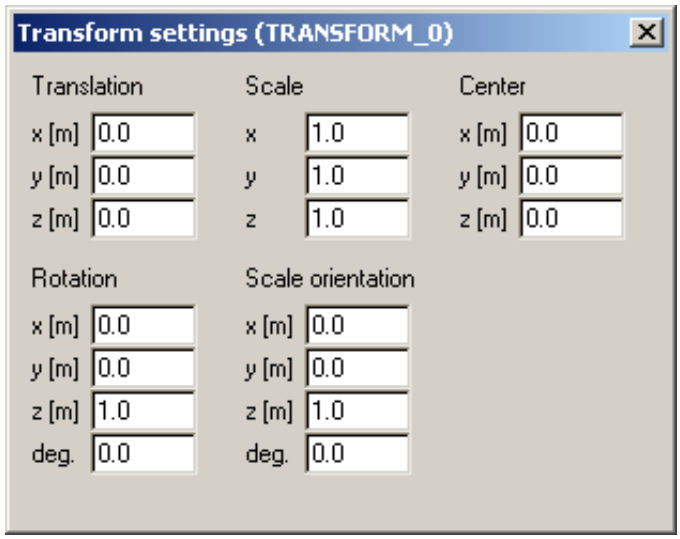
2.4. Transformation nodes

In the previous sections we learned how to create elementary shapes. Each shape was centred in the coordinate system: Boxes, spheres and cylinders were positioned with their centres of gravity situated at point (0, 0, 0), the centre of axis of cone was also situated at that point. If we created two or more shapes they were overlapping.

It's time to learn how to move the objects within the coordinate system, how to rotate them or to change their scale. These operations are performed using the so-called transformation node. You can think of it as a repository which can hold other nodes inside.

Select `ROOT` in the nodes list and click New transform button in the main panel. After entering a node name the Transform settings dialog opens:

Figure 2-8. Transform settings dialog



To see the effects of transformation node we place a box inside it. Select the transformation node in nodes list and create new box using New shape. We will use this procedure extensively in the successive sections, you should therefore remember that to insert a new node into an existing tranformation node, you first select the transformation node in the nodes list and then create the new node.

Now let's return to the Transform settings dialog. The first parameter is the translation vector. Its three components are again measured in metres. Type `1` in the x textfield and confirm it with Enter - the box moves 1 metre along the x axis, i.e. to the right in the default view. You should try to move it along the y and z axes to practice using the coordinate system.

After you set the translation back to zero we'll examine rotation. Using this parameter it is possible to rotate objects around an arbitrary line passing through the origin. Line direction is specified as a vector which you enter into the x, y, z textfields. If we want to rotate our cube around the y axis, we use the `0 1 0` vector. You then type the angle in degrees (e.g. `30`) into the last textfield and confirm it with Enter. You can use both positive and negative real numbers to rotate the box in either direction.

Rotation around an axis which is not passing through the origin is possible using the center parameter. Here you enter coordinates of the point the line should be passing through.

We turn the box back to the initial orientation and look at the scale parameter. It is possible to change the scale of objects independently in the direction of all three coordinate axes. If we set the x component of the scale vector to 2, the cube becomes a block with double width. However, we could have achieved the same effect by setting the box width to 2 metres. It is more interesting to use scale with other shapes, e.g. a sphere can be deformed into an ellipsoid.

It may happen that we don't like changing the scale along the current coordinate axes. For the purpose of the scale operation the coordinate system may be rotated to get new x, y, z axes along which the scaling takes place. The components of the scale orientation parameter have the same meaning as the rotation components introduced earlier, i.e. the first three specify a directional vector and the last one is the rotation angle. In this way we can e.g. deform a box into a parallelepiped.

What happens if we specify translation, rotation and scale all at once? We know that the result is dependent on the order of these transformations. VRML prescribes the following fixed order:

1. scale
2. rotation

3. translation

Let's practice it with our cube example: Enter a double scale in the x-axis direction and a `20` degrees rotation about vector `0 1 0`. The result is a rotated block.

What do we do if we want to perform the rotation first? We have to use two transformation nodes: Start with an empty scene (click Delete all). Create a transformation node called `SCALE` (select `ROOT`, click New transform). Place another transformation node called `ROTATION` inside `SCALE` (select `SCALE`, click New transform). Add a box inside `ROTATION` (select `ROTATION`, click New shape). In the `ROTATION` node dialog enter `20` degrees rotation around vector `0 1 0`. In the `SCALE` node dialog set the y component of scale to `2` and that's it.

We see that the transformations add together from descendants to ancestors: A cube is placed inside `ROTATION`, the result is a rotated cube. This object is inside `SCALE` and the result is a parallelepiped.

> **Note:** The nodes list displays nested nodes in such a way that names of all nodes inside a transformation node are right indented from it.
>
> All nodes are descendants of the `ROOT` transformation node, which represents an identity transformation. Each time we add a node into an existing transformation node we first select the transformation node's name. In case we don't want to apply any transformations on the new node we place it inside the `ROOT` node. The `ROOT` node cannot be deleted.

2.5. Putting VRML and Dynast together

We have already covered all topics to be able to create simple VRML models. This section explains how to add motion to the static models - we link them with Dynast simulations. The procedure will be shown using a simple example: We create a model of door which close automatically after being opened.

We start with the VRML model. The door itself would be represented by a box. Since we want to turn the door around a hinge we place the box inside a transformation node.

Select `ROOT` and click New transform to create a node called `ROTATION`. Leave the Transform settings dialog open. Select the `T_DOOR` node and click New shape to create a node called `DOOR`. In the Shape settings dialog we select box and make it 1metre wide, 2 metres high and 0.05 metre deep. A realistic appearance can be achieved using a texture - you may use e.g. the image from `http://as.matfyz.cz/examples/door.png` (enter the address into the Texture URL textfield in Appearance section. We can furthermore brighten the texture by setting emissive color to `0.3 0.3 0.2` (it is better to zoom the door in when setting the appearance, e.g. using the Fit button in Cortona).

Close the Shape settings dialog and return to `T_DOOR` node. Since we want the door to be ajar at the beginning of the simulation, we have to rotate them around a line parallel with the y axis which passes through point -0.5 0 0 (the door is 1 metre wide). We therefore set center to `-0.5 0 0` and the first three components of rotation to `0 1 0`. We choose an angle of `50` degrees.

For a better impression we add the door hinge represented by a cylinder. We have to move it to the left border of the door. Select `ROOT` and click New transform to create a node called `T_HINGE`. Set the translation vector in Transform settings dialog to `-0.5 0 0`. Then select the `T_HINGE` and click New shape to create node called `HINGE`. In the Shape settings dialog select cylinder and make it 2.2 metre high with 0.02 metre radius. Go to Appearance section and select a brown colour by setting diffuse color to 0.4, 0.2, 0.

The model is now finished and looks like this:

Figure 2-9. Door model

Now we have to assemble a mathematical description of the simulation in form of a Dynast script. Suppose the door is moving according to the differential equation

$$\alpha'' + m\alpha' + k\alpha = 0$$

(where α is the door angle measured from equilibrium position), which is the well-known equation for damped oscillations. We choose e.g. `m`=0.5 a `k`=1.

Click the Dynast script button in the main panel - there appears a new window where you type the following script:

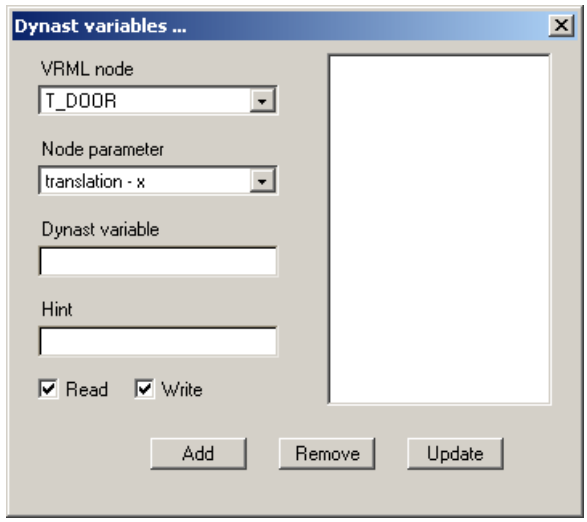
```
*SYSTEM;
k = 1;
m = .5;
SYSVAR alpha, beta;
0=beta-VD.alpha;
0=VD.beta+m*beta+k*alpha;
*TR;
TR 0 10 infinite;
INIT alpha = 0.873;
RUN;
*END;
```

> **Note:** We assume here that the reader is already familiar with the Dynast input language, otherwise he is encouraged to have a look in the Dynast user manual available e.g. at http://virtual.cvut.cz. On the other hand the example presented above is almost self-explanatory, we just mention that `VD.` stands for a time derivative.

> **Tip:** The initial value of variable `alpha` in the `INIT` section corresponds to 50 degrees converted to radians. The angles in VRML are measured in radians even if our editor uses degrees for the user's convenience. The arguments of e.g. trignometric functions in Dynast are entered in radians, too.

Close the Dynast script dialog after you finished typing. We now link the variable `alpha` to the parameter `rotation` of node `T_DOOR`, i.e. we want that each time the value of `alpha` changes it should be copied into the last component of the `rotation` parameter (this is the component which corresponds to the current door angle). Select `T_DOOR` and click the Variables button in the main panel. The Dynast variables dialog appears:

Figure 2-10. Dynast variables dialog



Here we set the connection between VRML parameters and Dynast variables. We see that the VRML Node list is already open on `T_DOOR` node (this is because we selected it in the nodes list before). Select rotation - angle in the Node parameter list. Enter the identifier `alpha` in the Dynast variable textfield and click Add. The link between `alpha` and `T_DOOR.rotation` has been created and you can close the dialog using the button in the upper-right corner.

> **Tip:** An existing link between a Dynast variable and a VRML node can be deleted by selecting it in list (right part of the panel) and clicking Delete. In case you want to change link's parameters select it again in the list, perform the necessary changes and click Update.

We now save the simulation to our Dyncad server account (we created it in the first section). Click Save / Load and enter your login and password. Fill in the Simulation name, e.g. `door`. Click Save to store the simulation and View to open it in the demodyn applet.

> **Note:** A single simulation consist of four files - VRML scene, Dynast script, configuration file with data from Dynast variables dialog and a HTML page with references to the previous three files and the demodyn applet. If you want to see the VRML file text before saving click the VRML source button in the main panel. You can also use the Config. file button to view the contents of configuration file. This may be helpful in case you use the online version of the editor but you want to save these files using a clipboard to your local disk.

2.6. Using clipboard and links

Most computer applications implement the so-called clipboard operations, which comprise *copy*, *cut* and *paste*. We find the corresponding three buttons in our VRML editor, too. They operate on nodes: Copy stores the selected node in clipboard, Cut does the same thing and deletes the original node, and Paste inserts the node from clipboard to the scene.

What happens to transformation node's children if we use the *cut* (or *copy*) operation on it? They are moved (or copied) into the clipboard, too. Consider for example the following scene:

```
ROOT
 TRANSFORM_1
  TRANSFORM_2
   SHAPE_1
```

Assume you forgot to insert a transformation node called `TRANSFORM_0` between `ROOT` and `TRANSFORM_1`. You select `TRANSFORM_1` and click Cut. Then select `ROOT` and create the `TRANSFORM_0` node. Select it and click Paste. Here is the result:

```
ROOT
 TRANSFORM_0
  TRANSFORM_1
   TRANSFORM_2
    SHAPE_1
```

The copy operation can be useful in case you need two or more objects which differ only in colour. You make as many copies as you want and set their appearance individually.

> **Note:** To avoid the situation when a node and its copy have the same names (which is forbidden) the editor adds a `COPY` suffix followed by the copy number to the original node name.

Another situation arises if we need two or more identical copies of an object. In this case we use so-called *link*. Links have the property that each time you modify the original node, the copy changes, too. For example when modelling a car you usually want the four wheels to be exactly the same. We practice links on a simple carriage model.

Assemble a scene according to the following scheme:

```
ROOT
 WHEELFRAME
 T_1
  WHEEL
 T_2
 T_3
 T_4
```

- `WHEELFRAME` - box with dimensions (1, 0.1, 0.3)
- `T_1` - transformation node with translation (0.375, 0, 0.25) and 90 degrees rotation around (1, 0, 0)
- `WHEEL` - cylinder with radius 0.15 and height 0.1
- `T_2` - transformation node with translation (0.375, 0, -0.25) and 90 degrees rotation around (1, 0, 0)
- `T_3` - transformation node with translation (-0.375, 0, 0.25) and 90 degrees rotation around (1, 0, 0)
- `T_4` - transformation node with translation (-0.375, 0, -0.25) and 90 degrees rotation around (1, 0, 0)

We insert the other three wheels into `T_2`, `T_3` a `T_4` using links. Select `T_2` and click New link in the main panel. Select `WHEEL` node in the dialog. After clicking OK the wheel copy appears. Insert two more links to `WHEEL` into nodes `T_3` and `T_4`.

Now try changing the color of the original `WHEEL` node. Observe that the link's colour changes, too.

> **Tip:** Link nodes cannot be edited, you might change only the original node. You recognize links in the nodes list as they end with the `link` suffix folowed by a link number. Each time you delete a node the editor also deletes all links to that node (since they are no more valid).

2.7. Viewpoints

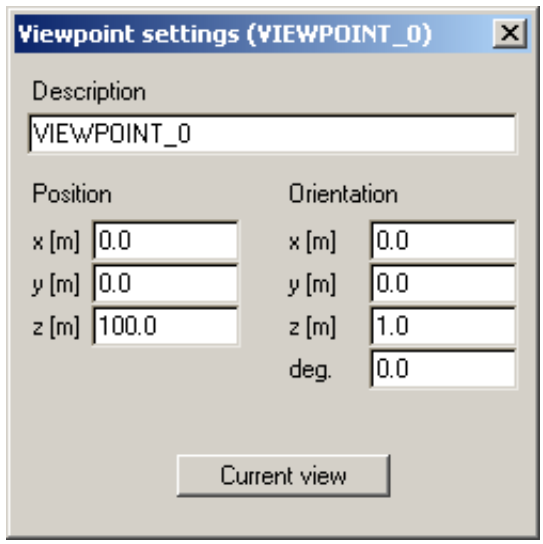
The user's default position in VRML is at point (0, 0, 10) and he looks towards the coordinate system origin. He can of course move, however, the authors of VRML scenes sometimes select interesting sites and offer the user to switch between them. This can be achieved using the so-called Viewpoint nodes. In this section we build a small-scale model of Moon orbiting round the Earth and use the Viewpoint node to set a nice view. We suppose that the reader is already familiar with creating Transform and Shape nodes so that we can describe them more briefly.

We'll represent Earth and Moon by spheres with textures. We insert both spheres into transformation nodes to be able to move them around.

Create a transformation node called `T_EARTH` and insert a sphere called `EARTH` inside it. Leave the parameters of `T_EARTH` without change. Set the Earth's radius to 6.37 metre and cover it with a texture, e.g. from `http://as.matfyz.cz/examples/earth.jpg` (the Earth is too big and disappears from the screen - to be able to see it click the Fit button in Cortona). Create second transformation node called `T_MOON` (insert it into `ROOT`, not `T_EARTH`!) and place a sphere called `MOON` with radius of 1.38 metre inside it. You can use the texture from `http://as.matfyz.cz/examples/moon.jpg`. Move the Moon by setting the x component of translation in `T_MOON` to 33.63 metre. The true distance is ten times greater, however, the scene wouldn't look good (you can try it, use Fit again).

Now let's use the Viewpoint node to select a nice view. Select `ROOT` and click New viewpoint in the main panel. You can leave the default name `VIEWPOINT_0`. The Viewpoint settings dialog appears:

Figure 2-11. Viewpoint settings dialog



It is possible to enter the viewpoint's description, position and view orientation. We already know that the default view is against the direction of z axis. You specify another view using the orientation parameter. Its four components have the same meaning as the rotation parameter in case of transformation node. E.g. the values 0, 1, 0 and 90 tell the browser to rotate the view by 90 degrees around the y axis and the new view direction is therefore parallel with the x axis.

Let's return to our simulation. Set the z-component of position vector to 100 metres and confirm with Enter. The editor moves to the new viewpoint so that we can check it. The VRML model is now finished and looks like this:

Figure 2-12. Moon and Earth simulation

1. It is possible to set the viewpoint not only by typing numbers. Move to the selected place directly in Cortona, and then click Current view in the Viewpoint settings dialog. The position and orientation will be set automatically.
2. If we create more than one viewpoint, the user can switch between them using the Next viewpoint and Previous viewpoint buttons in Cortona (or using the context menu).

Here is the script for Dynast (enter it in the Dynast script dialog):

```
*SYSTEM;
m=5.976*(10**12);
k=6.67*(10**(-11));
r_e=6.37;
r_min=33.63;
r_max=40.55;
a=(r_min+r_max)/2;
g=9.81;

SYSVAR x, z, vx, vz;
0=vx-VD.x;
0=vz-VD.z;
r=sqrt(x**2+z**2);
0=VD.vx+k*m*x/r**3;
0=VD.vz+k*m*z/r**3;

*TR;
TR 0 10 infinite;
INIT x=r_min, z=0, vx=0, vz=r_e*sqrt(g*r_max/(a*r_min));
RUN;
*END;
```

These equations express the Newton's laws of gravity and motion. The Moon orbits round the Earth in the xz plane and we neglect the influence of Moon to Earth.

The `x` and `z` variables correspond to the Moon's position; we link them with the translation parameter of node `T_MOON`. Select `T_MOON` and open the Dynast variables dialog (click Variables). The translation - x parameter of node `T_MOON` is already selected; type `x` in the Dynast variable textfield and press Add. Then choose translation - z in Node parameter list, type `z` into Dynast variable and click Add again. Close the dialog window.

Save the simulation in Save / Load dialog (e.g. under the name `earth-moon`) and click View to see the result.

2.8. Prototypes

Apart from the standard nodes (such Shape or Transform) the VRML enables user to define so-called prototypes. You can imagine that we take an ordinary VRML scene and assign it a name. The new name acts as an ordinary VRML node name but represents the whole scene. A prototype can also have several parameters which correspond to selected parameters of nodes inside the scene. Prototype definitions are stored in ordinary VRML files. We can use prototypes in another files by referring to the file with prototype definition.

A single file can define more than one prototype - in this way it is possible to create libraries of specialized objects (e.g. an interior designer can have different models of tables, chairs etc. at his disposal).

Our VRML editor includes a few prototypes which can be used immediately. Other prototypes might be imported. We'll demonstrate work with prototypes on a simple simulation: We create a road with two cars. The user sets initial velocity and acceleration for both of them and watches which car arrives first to the destination. We also want to display the distance mastered by both cars.

Create new transformation node called `T_ROAD`. Insert a box called `ROAD` into it and set its dimensions to (60 m, 0.01 m, 10 m). Improve the road appearance, e.g. set diffuse color to (0, 0.6, 0.6). Set the x-component of translation in `T_ROAD` to 25 metres so that the road starts approximately in the coordinate system origin.

Next we import the car prototype from `http://as.matfyz.cz/examples/car.wrl`. We have to tell the editor the prototype declaration which looks like this:

```
EXTERNPROTO Car
[ exposedField SFColor color ]
"http://as.matfyz.cz/examples/car.wrl"
```

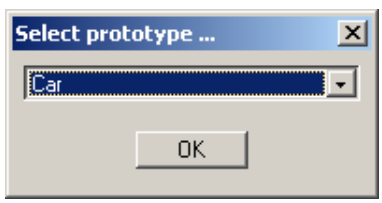
Each declaration starts with the `EXTERNPROTO` keyword. Then follows the prototype name and a list of its parameters in brackets. It ends with the prototype URL enclosed in quotes. You can always obtain the protype declaration (and the description of its parameters) from the author of the prototype.

> **Note:** Again you can use both local prototypes and those available on Internet. If you work with the online version of VRML editor use only prototypes available from Internet.

Click Import VRML button in the main panel and insert the prototype declaration listed above. After clicking Load the prototype is ready to use.

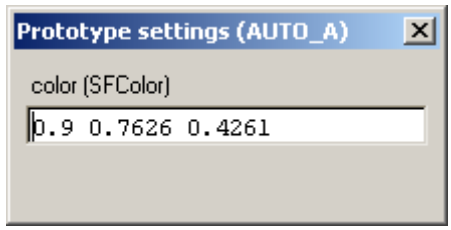
Since we want our cars to move, we insert them in transformation nodes. Create a transformation node called `TRANSFORM_A` (under `ROOT`) and insert the car prototype. This is done as follows: Select `TRANSFORM_A` and click New prototype button. Select Car from the list.

Figure 2-13. Selecting new prototype



Prototypes are nodes and must be assigned a name. We will call this prototype `CAR_A`. The car appears together with the settings dialog:

Figure 2-14. Car prototype settings



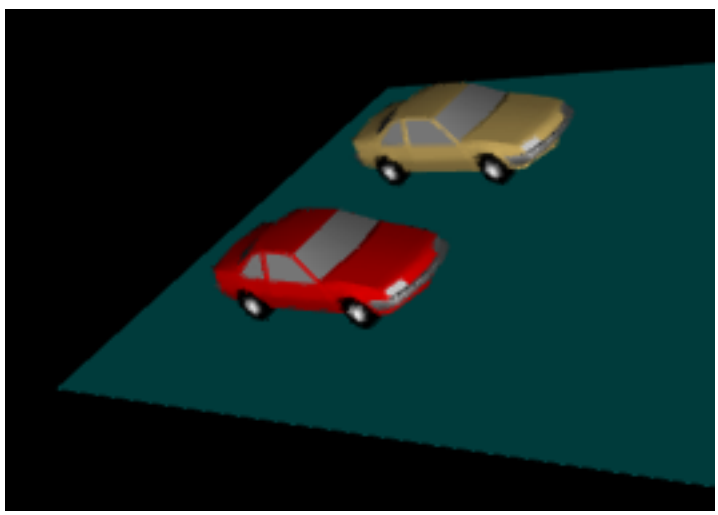
The *Car* prototype has a single parameter called color. You enter new values of color as a triple of R, G, B values from [0, 1] (the same as in case of *Shape* nodes) separated with spaces. Let's use red, i.e. type `1 0 0` in the textfield and confirm it with Enter. You can close the dialog now.

> **Note:** The editor doesn't check whether the entered values are correct. In case you make a syntactical mistake there appears a window with Cortona error message. Close the window and fix the parameter values.

We now return to `TRANSFORM_A` node. Set the z-component of its translation to -2 metres - this moves the car to the right side of the road.

Now create the second car in the same way: Add a prototype called `CAR_B` inside a transformation node `TRANSFORM_B`. Set the z-component of translation to 2 metres. The scene now looks like this:

Figure 2-15. Scene with road and two cars

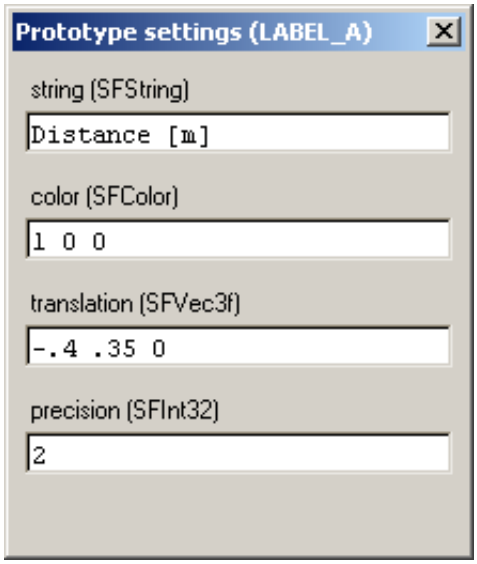


> **Tip:** The scene does't look well from the default viewpoint. You might add a Viewpoint node to select a nice view of the whole road.

We promised to show the distances travelled by both cars during the simulation. There is a prototype called *Label* which is useful for displaying numerical values. It is already shipped with the editor and you don't need to import it.

Select ROOT, click New prototype and choose Label. We'll call the prototype LABEL_A.

Figure 2-16. Label prototype settings



The prototype has a number of parameters:

- string - this text will be displayed above the numerical value. Type `Distance [m]` in the corresponding textfield
- color - text colour. Select red (`1 0 0`) so that everybody knows it is the distance travelled by the red car.
- translation - text position on the screen. It is a triple of numbers - the first moves text left or right, the second up or down, the third forward or backward (i.e. it increases or decreases the text size). We have to guess the right values by trial, usually from interval [-1, 1]. We enter `-.4 .35 0` to move the text to the upper-left corner.
- precision - number of digits after the decimal point, i.e. the precision of displayed numerical value. We leave the default value `2`.

1. The text displayed using the *Label* prototype always stays in the same position even if the user moves. In addition it is not influenced by transformation nodes. The only way to set the text position is to use its own translation parameter.
2. The prototype has one more parameter called value - it is the numerical value being displayed. This parameter cannot be set in the dialog, you have to link it with a Dynast variable.

In the same way create a second *Label* prototype called LABEL_B which will display the distance travelled by the second car. Set its translation to `.15 .35 0` to move the text to the upper-right corner.

Enter the following Dynast script:

```
*SYSTEM;
aA = 0;
aB = 1;
length=50;
SYSVAR xA,vA,xB,vB;
```

```
0=VD.xA-(xA<length)*vA;
0=VD.xB-(xB<length)*vB;
0=VD.vA-aA;
0=VD.vB-aB;

*TR;
TR 0 15 infinite;
INIT vA = 4, vB = 0, xA = 0, xB = 0;
RUN;
*END;
```

The variables `xA` and `xB` contain the current car positions, `vA` and `vB` correspond to their velocities. The constants `aA` and `aB` are accelerations for both cars. Do not forget the spaces at both sides of equality sign when setting the initial values of the mentioned variables - the reason will be explained later.

Open the Dynast variables dialog and link the variable `xA` with the x-component of `translation` in `TRANSFORM_A`. Link the same variable with the `value` parameter of node `LABEL_A`. In the same way link the variable `xB` with the x-component of `TRANSFORM_B.translation` and with `LABEL_B.value`.

Save the simulation and check it in the demodyn applet. Here we can modify the initial positions, velocities and accelerations in the text of the script (you have to restart simulation each time you make a change). However, the user has to study the script first to grasp the meaning of individual variables. It would be better if we displayed a list of variables which can be modified together with a short description.

To do this we return to the VRML editor and open the Dynast variables dialog again. The Hint textfield shown here can be used to enter the description of a variable in the current link. Select the link between variable xA and node `TRANSFORM_A` which we made before. Enter the text `Car A initial position [0 - 50 m]` into the Hint textfield and click Update. Then select the link between xB and `TRANSFORM_B`, enter `Car B initial position [0 - 50 m]` into Hint and click Update again.

After opening the simulation in demodyn we see the two textfields:

Figure 2-17. Initial conditions in demodyn applet

What happens if we enter the value 10 in the first textfield and confirm it with Enter? The applet copies that value into `TRANSFORM_A` (the car moves forward) and replaces the assignment `xA = 0` in Dynast script by `xA = 10`. It is important to know that it replaces only the assignments which contain a space at both sides of the equality sign. In this way we prevent it from replacing equations such as `xA=xB`.

We now explain how to create textfields for setting the initial velocity and acceleration. We do it again in the Dynast variables dialog. We cannot easily repeat the procedure we performed with `xA` and `xB` since there is no VRML node linked to `vA`, `vB`, `aA` or `aB`. For this purpose there is a special `null` item in the VRML node list which we select. We now proceed as before and enter `vA` in the Dynast variable textfield. Then type `Car A initial velocity [m/s]` into the Hint textfield and click Add. Repeat the same procedure with the other three variables.

We end this section with two more tips concerning prototypes:

1. When saving VRML files the editor appends declarations of all prototypes we used in the current scene. There is one exception: If you save an empty scene the editor saves declarations of all prototypes which have been imported. In

this way you can save declarations of your favourite prototypes so you don't have to import them every time - you just load the empty scene instead.

2. In case you import a prototype and there is already a prototype with the same name in editor's memory its declaration will be overwritten. Make sure you have deleted all nodes inserted using the old declaration which is no longer valid.

2.9. Importing VRML nodes

Throughout this tutorial we used the *Shape*, *Transform* and *Viewpoint* nodes. In case these nodes don't fit our needs we can use another VRML node. However, we have to enter it directly as a VRML language text.

Assume we aren't satisfied with the black backround of our scene. It can be changed using the *Background* node, whose description is available e.g. in The Virtual Reality Modeling Language (http://www.web3d.org/technicalinfo/specifications/eai/index.html) specification.

Select `ROOT` and click Import VRML. Then insert the following text in the dialog:

```
Background
{
  skyColor [0 0 1, .6 .8 1, 1 1 1]
  skyAngle [1.5, 1.57]
}
```

New background appears after clicking Load - it now resembles a blue sky.

> **Note:** If you make a syntactical mistake there appears an error message from Cortona. Close the window, fix the error and click Load again.

We can edit imported nodes later in the usual way (by clicking Edit node or double-clicking in the nodes list).

> **Tip:** It is possible to import not only standard VRML nodes but also special nodes which are supported by Cortona.

If you enter two or more nodes in the Import VRML dialog at once, the editor recognizes them and imports them as individual nodes. You can even import complete VRML files! In case you have a VRML file exported e.g. from *3D Studio Max*, you can copy its text (using clipboard) to Import VRML dialog and then load it.

These limitations hold for importing VRML files:

- They cannot contain `PROTO` definitions. Loading files with `PROTO`s leads to unpredictable results.
- The `EXTERNPROTO` declarations are permitted, but they must all follow just after the `#VRML V2.0 utf8` header. Files with `EXTERNPROTO`s at different places lead to unpredictable results.
- The editor does not support `ROUTE` statements. It loads them correctly but they will be ignored when saving the file again.

> **Note:** Complicated scenes with a lot of nodes take some time to load (a few seconds). You find out the loading is finished after Import VRML dialog closes.

2.10. Using offline version

In the introduction we mentioned the security limitation which doesn't permit Java applets to write files to disk. It is possible to overcome this limitation in case of MS Internet Explorer browser: Go to the Tools menu, select Internet options and Security. Switch to Local intranet and click Custom level. Find the Java permission item, select Custom and click Java custom settings. Turn on all permissions for unsigned content.

Now download the file http://as.matfyz.cz/editor/editor_offline.zip and unpack it in a new directory. You get two subdirectories, `editor` and `demodyn`. See the user's guide to demodyn for more information about running it from local disk. Now go to `editor` subdirectory and open the file `editor.htm`. Look for the following two lines:

```
<PARAM name="mode" value="offline">
<PARAM name="base_directory" value="c:\">
```

The first already tells the editor to run in offline mode and you don't need to modify it. On the second line replace `c:\` with the directory where you unpacked `editor_offline.zip` (i.e. the directory with `editor` and `demodyn` subdirectories). Don't forget to include backslash at the end of the path! Save the modified file and open it with your web browser. You have just successfully started the offline version.

The difference between online and offline version is in the behaviour of Save / Load dialog. In offline version you don't need to enter any login or password. All simulations are stored in the `demodyn` directory - in this way they can include the demodyn applet. Use the Save button to save a simulation to `demodyn` directory and View to open it with demodyn. Use Load to open simulations in the editor or Delete to remove them from disk.

> **Tip:** In offline mode you may use local textures and prototypes as well as those accessible from Internet.